John von Neumann Institute for Computing

Parallelization of an Algorithm for Finding Facility Locations for an Entering Firm Under Delivered Pricing

J.L. Redondo, I. García, P.M. Ortigosa, B. Pelegrín, P. Fernández

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 269-276, 2006.

http://www.fz-juelich.de/nic-series/volume33

Parallelization of an algorithm for finding facility locations for an entering firm under delivered pricing

J.L. Redondo[a], I. García [a], P.M. Ortigosa [a], B. Pelegrín [b], P. Fernández[b]

[a]Dpt. Computer Architecture and Electronics, University of Almería, Spain.

[b] Dpt. Statistics and Operational Research, University of Murcia, Spain.

Abstract

This work presents the parallelization of an algorithm for finding facility locations for an entering firm which has to make decisions on the locations of its facilities as well as on its price setting in order to maximize profit. This combinatorial location problem is solved by *GASUB*, a new multimodal genetic algorithm with subpopulation support. The high computational requirements of the location problem demands the parallelization of the method. In this work two standard strategies have been implemented and compared. The first one follows a *master-slave* model, and the second strategy is a *coarse-grain* parallelization.

Keywords: Competitive location, Global optimization, Evolutionary computing and genetic algorithms, Parallel strategies

1. Introduction

Location decisions are frequently made by an entering firm that has to compete for customers with other pre-existing firms in the market. This has led to a variety of locations models with the aim of finding strategic locations for profit maximization (see [1]). These models are often extremely difficult to solve, at least optimally. Even the most basic models are computationally intractable for large problems instances, as it happens when the firm has to select a number s of locations in a set of m potential sites. In this case, the firm has to explore a large number ($\frac{m!}{s!(m-s)!}$ combinations) of possible locations to find an optimal solution. This is a combinatorial optimization problem and is hard to solve for high values of m and s.

We are interested in solving the problem of finding facilities location for an entering firm under delivered pricing. In this location model, the firms offer to sell the product to customers and also provide delivery for a single bundled price. Then customers buy from the facility that offers the lowest price in the area they belong to. In this setting, the entering firm have to make strategic decisions on location and price for maximizing profit. As a result of price competition, it is shown that a price equilibrium exists for any set of fixed facility locations (see [3]). Then the location-price decision problem is reduced to a new location problem (see [2]).

This work describes a new genetic algorithm with subpopulation support (GASUB), which inherits some features of UEGO ([4,5]), a stochastic optimization algorithm. GASUB is a general algorithm for solving combinatorial optimization problems which has proved its capabilities for finding optimal solutions for the location problem. In [6] GASUB is evaluated and compared to one of the standard software tools (Xpress-MP [7]) frequently used to deal with this kind of problems. It has been shown that GASUB can compete with Xpress-MP in finding global optima. For very hard problems Xpress-MP was not able to reach a solution while GASUB was. However, when the number of new locations increases, the execution time for GASUB becomes unacceptable, so parallel implementations can help to eliminate this drawback.

GASUB was designed with parallelism in mind, so its parallel implementations do not need complicated parallel models but the simple master-slave or the coarse grain strategies are good enough to obtain efficient solutions for problems of a very high computational cost. This work explores and evaluates two parallel implementations of GASUB with application to complex location problems.

In the following sections more details of this interdisciplinary problem will be provided. The problem of finding facilities location for an entering firm is described in Section 2. Section 3 contains a short description of GASUB, the genetic optimization algorithm. Details of our parallel implementations of GASUB and their evaluations on a cluster of processors are the topics of Sections 4 and 5, respectively. Finally, Section 6 draws some conclusions.

2. The entering firm location-price problem

We consider a set $C = \{1, 2, ..., n\}$ of spatially separated market areas within a region R. Customers in area i are aggregated at a market point v_i in R. Demand for a homogeneous product is assumed to be known and fixed, being w_i the amount of product required at v_i.

The product will be served by a set of facilities $F = \{1, 2, ..., \bar{q}\}$, the first q ($1 < q < \bar{q}$) already exist in the region and the others $s = \bar{q} - q$ are to be located. Each facility j is located at a point f_j in R. The points f_1 to f_q are known and the points f_{q+1} to f_m have to be selected in a set L of locations, which contains m potential sites. All the market and the location points are nodes in a transportation network immerse in R, where the product will be delivered. With d_{ij} we denote the distance between points v_i and f_j, the transportation cost per unit of product and unit of distance is denoted by t. Each facility j will deliver the product to customers in v_i at a unit price p_{ij} which include the transportation cost. It is assumed that all facilities have the same production cost p_{prod}, and that $p_{ij} \geq p_{min} + td_{ij}$, where p_{min} is the minimum price per unit of product (greater than the production cost).

Customers buy at the facility that offers the lowest price in the area they belong to. If two or more facilities set the lowest price at some market point v_i, customers in area i buy at the closest facility. Furthermore, since this assumption does not prevent that, when more than one of the cheapest facilities is also the closest, ties still may occur, we consider that half of the market in area i is captured by the new firm.

As a result of price competition (See [2] for an analysis on the process of price changing), the entering firm can only capture market at the points v_i such that $d_i^S \leq D_i$, where S denote the set of locations for the new facilities ($S \subset L$), $d_i^S = \min\{d_{ij} : f_j \in S\}$ and $D_i = \min\{d_{ij} : j = 1, ..., q\}$. The optimal price at v_i offered by the new firm is $p_i^S = p_{min} + tD_i$. Let $C_S^1 = \{i \in C : d_i^S < D_i\}$, $C_S^2 = \{i \in C : d_i^S = D_i\}$ and $p_{net} = p_{min} - p_{prod}$. Then, the maximum profit the new firm gets by locating its facilities in S is :

$$\Pi(S) = p_{net}\Big(\sum_{i \in C_S^1} w_i + \frac{1}{2}\sum_{i \in C_S^2} w_i\Big) + t\sum_{i \in C_S^1}(D_i - d_i^S)w_i$$

In order to define a search domain and the structure of the points defined in this domain, this combinatorial problem must be encoded. A point (individual in terms of genetic algorithms) consists of a single string that is a collection of m bits. The position of a bit in the string coincides with the index of the associated facility. Each bit can have 0 or 1 values, where 1 indicates that the associated facility has been chosen as part of a solution. As the set S of selected facilities is predetermined for every problem, the number of bits to 1 ratio must be fixed to the number of new facilities to be selected (cardinal of S). We must consider this constraint when generating any search point.

3. GASUB: A genetic algorithm for global optimization

GASUB is a genetic algorithm with subpopulation support, where every subpopulation is intended to occupy a local maximizer of the fitness function with no a priori knowledge of the total number of local optima in the domain function. This means that when the algorithm starts running it does not know how many subpopulations will appear, so it is necessary to have a mechanism to create and fuse subpopulations. Thus, new subpopulations are created when it is likely that the parents are on different hills, and subpopulations have to be fused when they are thought to climb the same hill.

To illustrate the way GASUB that copes with unevenly spread optima, it is natural to use a terminology that is well known from the field of simulated annealing. Thus, when illustrating our definitions and methods, we will talk about the 'temperature' of subpopulation, the ability of escaping from local optima. In our system, we made the 'temperature' an explicit attribute of every subpopulation (it is the attraction of subpopulations). This allowed us to offer an algorithm that 'cools down' the system while subpopulations of different 'temperatures' are allowed to exist at the same time. The basic idea of the algorithm is that 'cooler' subpopulations are allowed to create 'warmer' subpopulations by autonomously discovering their own local area of attraction.

A key notion in GASUB is that of a *subpopulation*. A particular subpopulation is not a fixed part of the search domain; it can move through the space as the search proceeds. A subpopulation would be equivalent to a single individual, which is defined by a center, a fitness function and a radius value. The center is a solution and the radius indicates the attraction area of this subpopulation (temperature). This definition assumes a *distance* defined over the search space. For our combinatorial problem we define the *Hamming distance*. Because of the constraint of the problem, the number of chosen facilities and hence the number of bits (or genes) to 1 is fixed, so the *Hamming distance* between any two feasible points (individuals) must be always multiple of 2.

The radius of a subpopulation is not arbitrary; it is taken from a list of decreasing radii, the *radius_list*. The radii decrease in a regular fashion in geometrical progression. The first element of this list is always the diameter of the search space (r_1), which will ensure that the largest subpopulation always contains the whole space independently of its center. The diameter is given by the largest distance between any two possible solutions according to the distance mentioned above, and it is an input parameter. If the radius of a subpopulation is the ith element of the list, then we say that the *level* of the subpopulation is i.

During the optimization process, a list of subpopulations is kept by GASUB and this *subp_list* defines the whole *population*. The maximal length of the *subp_list* is given by *max_subp_num*, which is an input parameter that indicates the maximum population size. The algorithm GASUB is in fact a method for managing the *subp_list* (i.e. generating, selecting and mutating subpopulations). See Algorithm 1.

At the *initialization* part a single subpopulation with a random individual as center is created. This individual must have the same ratio of genes to 1 as the number of new facilities and its associated radius will be equal to the diameter of the search space.

The *mutation* procedure applies consecutive interchanges of one facility to every center of every subpopulation. At the *generation* procedure, when it is known that more than one local maximum exists inside the subpopulation, each subpopulation in the list is divided in two or more.

The *selection* procedure has two mechanisms, the first one tries to fuse subpopulations that are too close, so if the centers of any pair of subpopulations from the *subp_list* are closer to each other than the radius associated to the current level, then these subpopulations are fused. The center of the new subpopulation will be the one with the best function value while the radius will be the largest one. The second mechanism selects the subpopulations to be maintained in the *subp_list*. If the

Algorithm 1 GASUB Genetic algorithm with subpopulation support

```
1 proc GASUB ≡
2   Initializing_population
3   Mutation(n_1)
4   for i := 1 to levels
5       Determine(r_i, new_i, n_i)
6       Generation_and_crossover(new_i/length(subp_list))
7       Selection(r_i, max_subp_num)
8       Mutation(n_i/max_subp_num)
9       Selection(r_i, max_subp_num)
```

number of subpopulations overcomes the allowed maximum number, the subpopulations that have been created more recently are eliminated.

GASUB is an iterative algorithm with *levels* iterations and with a budget determined by the maximum number of function evaluations (N). Both, *levels* and N are users-given parameters.

At each level, in addition to the radii (r_i), two important parameters are computed; the number of function evaluations for subpopulations generation (new_i) and the mutation (n_i). These principles are described in detail by [4,5].

4. Parallel strategies

The high computational requirements of the optimization algorithms have raised the appearance of numerous parallelization strategies. In this work two parallel strategies have been implemented, an asynchronous *master-slave* model and a *coarse-grain* model.

4.1. Asynchronous master-slave strategy

In the *master-slave* model (PAGASUB) the parallelism comes from the evaluation of the individuals in the population. This is due to the fitness of an individual being independent from the rest of the population, and there is no need to communicate. The evaluation of individuals is parallelized by assigning a fraction of the population to each available processor. Communication occurs only as each slave receives its subset of individuals for evaluation and when the slaves return the fitness values. If the algorithm stops and waits to receive the fitness values for the entire population before proceeding into the next generation, then the algorithm is synchronous. We have implemented an asynchronous *master-slave* algorithm where the algorithm does not stop to wait for any (slow) processors.

The *slaves* processors only need to receive the two features of any subpopulation (i.e. its center and its radius), from the *master* processor. In order to be able to run the *mutate* and *generate* procedures, so the amount of information involved in the communication procedures is quite small. Both procedures do not need any additional information; they depend only on a single subpopulation. For this reason, these procedures can be run independently.

At the initialization phase, the *master* processor creates the initial *subp_list* containing several individuals. Then it distributes the species among the *slave* processors so all subpopulations can be optimized simultaneously. When the slaves finish, they send the results to the *master* processor and it updates the list of subpopulations that will be used in the following level. Then an iterative process (*levels*) is carried out (See Figure 1).

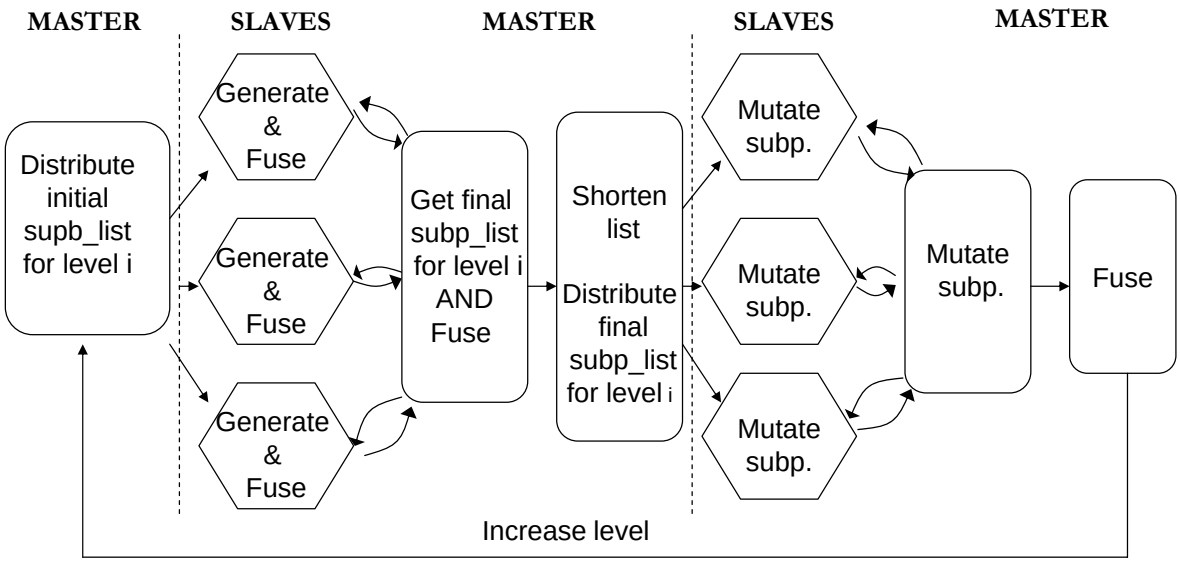

Figure 1. Asynchronous master-slave model

Rectangular and hexagonal boxes represent commands executed by the *master* and the *slaves* processors, respectively. Vertical dotted lines indicate synchronisation points and arrows represent communications.

Initially, the *master* processor distributes the *subp_list* among the *slave* processors. The *slave* processors pick up the subpopulations and evaluate them, trying to create new subpopulations. When the *slaves* finish the generation procedure, they fuse their list of subpopulations before sending them to the *master* processor.

In a synchronous version, the *master* processor stays in a wait state until **all** the *slave* processors finish generating subpopulations. Once the *master* processor has received all the new subpopulations, it applies the fusion and selection processes to them in order to complete the final *subp_list* at this level. Later, it distributes this list among the *slave* processors, which take charge of mutating each of their assigned species. When all species have been mutated, they are sent to the *master* processor that applies a fusion process and forms the *subp_list* that will be used in the following iteration.

In this case, the distribution of the computational load is not well balanced. The evaluation of the objective function is carried out within the subpopulation generation and mutation processes, which are only executed by the *slave* processors. Therefore, the *master* processor is mostly waiting for results from the *slave* processors. Due to the fact that the performance of this synchronous version is very low, elimination of some of synchronisation points is necessary. In this way the *master* processor can also work on the generation and mutation processes and the processors can distribute the computational load in a more dynamic way.

In the asynchronous implementation (PAGASUB) solves some of the previous problems. In this implementation the load has been balanced forcing the *master* processor to mutate and generate subpopulations while the *slave* processors are working. Now the *master* processor can start to carry out the synchronous operations over the species before it has received all the information, so the idle time can be reduced considerably.

Algorithm 2 CGGASUB Coarse-grain model

```
1  proc CGGASUB ≡
2    Initializing_population
3    Mutation(n_1)
4    for i := 1 to levels
5      Determine(r_i, new_i, n_i)
6      Generation_and_crossover(new_i/length(subp_list))
7      Selection(r_i, max_subp_num)
8      Mutation(n_i/max_subp_num)
9      Selection(r_i, max_subp_num)
10     if (i > levels/2)
11       Migrate_subpopulation
```

The *master* processor is constantly checking for the arrival of information (a new generated sublist of subpopulations or a mutated subpopulation) from the *slave* processors and if any, it immediately sends back a new species. Otherwise the *master* processor contributes to the optimization process by fusing or mutating the received subpopulations. These processes executed by the *master* processor are always interrupted when new information arrives from any *slave* processor. When all subpopulations have been mutated, the master processor applies a fusion process and creates the subpopulation list that will be used in the following iteration.

4.2. Coarse-grain strategy

In the *coarse-grain* model (CGGASUB), each processor executes the GASUB algorithm on different subpopulations *subp_list* in an independent way. Nevertheless, intermediate results (subpopulations) are sometimes exchanged among processors. The *coarse-grain* structure is shown in the Algorithm 2.

In the *Migrate_subpopulation* process, each processor sends all other processors the subpopulation with the best fitness or function value. This migration process is only carried out at the half last levels. Migration is not carried out at early phases of the algorithm in order to allow the populations to evolve independently. Taking into account that establishing all to all communications (broadcast) is very expensive when the number of processors raises, the migration has been implemented using a recollector processor, which receives the individuals from the remaining processors, joins them into a sublist and sends this to all processors.

In this parallel implementation, only one of the input parameters is modified regarding to the sequential version: For each computer, the size of *subp_list* is obtained by dividing the initial list size between the number of processors.

5. Experiments

The parallel algorithm has been implemented using C++ and the message passing library MPI. The experiments have been carried out on a cluster, which consists of 52 900 MHz Ultrasparc III processors in a single cabinet with 1 Gb of memory per processor, totally 52 GB. The level 1 cache on the UltraSPARC-III is 64 KB, 4-way set-associative with 32 byte lines. The level 2 cache on the UltraSPARC-III is 8 MB, direct-mapped with 64 byte lines.

For computational experiments we have selected 1046 cities as nodes, and their populations as

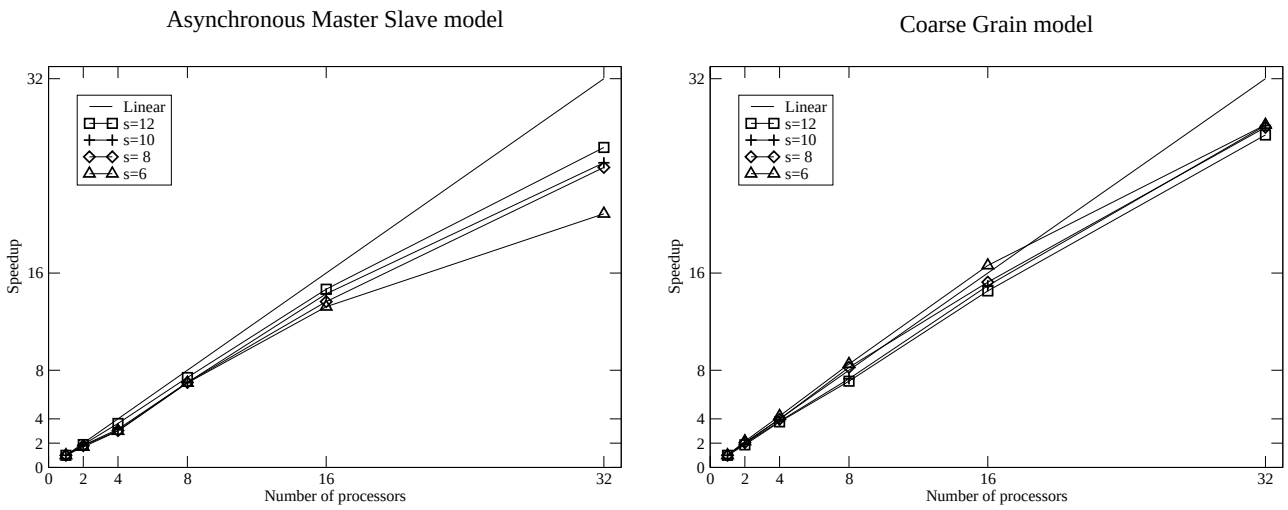

Figure 2. Speed up Asynchronous master-slave model (left) Coarse-grain model (right)

demand. The number of pre-existing facilities was equal to $q = 10$, the number of new facilities were equal to $s = 6, 8, 10, 12$, the unit transportation cost was equal to $t = 0.001$. Finally, we fixed $p_{net} = 1$. More details about the features of these experiments are shown in [6], where results from the sequential version are compared to results obtained when solving the problem with Xpress-MP, an integer linear programming optimizer. In this section these experiments have been executed and evaluated by the parallel versions. The experiments were executed using P=2^i processors, with $i = 0, 1, 2, 3, 4, 5$.

Due to the stochastic nature of the algorithms, all results given in this work are average values of ten executions, obtaining a statistic ensemble of experiments. In order to measure the reliability and efficiency of the parallel algorithms, the obtained results have been compared to the sequential ones. First, the convergence to optimal solutions with the same precision than sequential version was verified, so the quality of the algorithms remains in the parallel versions. The number of evaluations in the asynchronous master-slave parallel version are similar to the sequential ones. However, in the coarse-grain technique the number of evaluations decreases lightly, due to the user given parameters approximation.

The execution time when varying the number of processors has been also computed, and speedup analysis carried out. Figure 2 shows the efficiency of the asynchronous master-slave and coarse-grain models.

In the *master-slave* model the number of communications depends on the complexity of the problem and stage of the algorithm, while in the *coarse-grain* parallel algorithm is fixed and proportional to the number of processors and the *levels* parameter. In this way, although both strategies do the same amount of work (number of evaluations), the communication cost increases the total execution time in the master-slave model. Consequently the *coarse-grain* model obtains better values for the speedup than the *master-slave* model.

In the coarse grained decompositions there are some cases where super linear speedup are obtained. This is due to number of function evaluations decreasing. With respect to the communication costs, in this model increasing the complexity of the problem (from s=6 to s=12) causes more uneven loading of the processors, and thus the communication takes longer (See Figure 2 (right)).

Nevertheless, in the *master-slave* model, even though the amount of work each process does

increases (from s=6 to s=12), the distribution of the load is well balanced and the communication costs do not increase as a result of complexity increment of the problem. So, the communication time is compensated by the computational time, improving the obtained speedup, as we can see in Figure 2 (left).

6. Conclusion

In this work we have proposed a new genetic-like algorithm, for finding solutions to different facility locations problems which are hard to solve. Taking into account the intrinsic parallelism of the algorithm, two standard strategies have been implemented and evaluated. Both strategies obtain the same quality of results and their corresponding percentage of success on finding the global solution is 100%.

The asynchronous master-slave model obtains a good speedup thanks to the dynamic distribution of the load and the participation of the master processor in the tasks with more computational complexity. The course-grain model obtains good speedup thanks to the intrinsic parallelism of the sequential algorithm. However, a light load imbalance occurs causing some uneveness in communication costs.

7. Acknowledgments

This research has been supported by the Ministry of Science and Technology of Spain under the research projects BEC2002-01026 and CICYT-TIC2002-00228, in part financed by the European Regional Development Fund (ERDF). This work was carried out under the HPC-EUROPA project (RII3-CT-2003-506079), with the support of the European Community - Research Infrastructure Action under the FP6 "Structuring the European Research Area" Program).

References

[1] H.A. Eiselt, G. Laporte, and J.F. Thisse. Competitive location models: a framework and bibliography. *Transportation Science*, 27:44–54, 1993.

[2] P. Fernández, B. Pelegrín, M.D. García, and P. Peeters. A discrete long-term location-price problem under the assumption of discriminatory pricing: Formulations and parametric analysis. *European Journal of Operations Research*, pages –, 2005. forthcoming.

[3] M.D. García, P. Fernández, and B. Pelegrín. On price competition in location-price models with spatially separated markets. *TOP*, 12(2):351–374, 2004.

[4] M. Jelásity, P.M. Ortigosa, and I. García. UEGO, an abstract clustering technique for multimodal global optimization. *Journal of Heuristics*, 7(3):215–233, 2001.

[5] P.M. Ortigosa, I. García, and M. Jelasity. Reliability and performance of UEGO, a clustering-based global optimizer. *Journal of Global Optimization*, 19(3):265–289, 2001.

[6] B. Pelegrín, P. Fernández, J. L. Redondo, I. García, and P. M. Ortigosa. Finding locations for an entering firm under delivered pricing competition. http://www.ace.ual.es/Investigacion/papers/05/report11.pdf. Technical Report TR-11, Departamento de Arquitectura de Computadores y Electrónica, Universidad de Almería, 2005.

[7] Xpress-MP. *Dashoptimization*, 2004.